CRIKA. Quand les regles rencontrent les schemas

François Rechenmann, Philippe Vignard

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports de Recherche

N° 468

CRIKA
QUAND LES REGLES RENCONTRENT LES SCHEMAS

François RECHENMANN
Philippe VIGNARD

Décembre 1985

CRIKA
QUAND LES REGLES RENCONTRENT LES SCHEMAS

François Rechenmann +
Philippe Vignard +

+ INRIA Sophia Antipolis
Route des Lucioles
06560 Valbonne

Tel 93 65 78 75

RESUME

Ce document décrit CRIKA, un logiciel d'Intelligence Artificielle permettant l'utilisation combinée de règles de production et de schémas.
On conserve les facilités d'utilisation et d'expression des connaissances de l'outil de base CRIQUET. Les schémas utilisés sont proches des frames mais permettent une représentation plus déclarative et uniforme.

ABSTRACT

This report describes CRIKA, an Artificial Intelligence system allowing to manipulate production rules and object based representation.
All the facilities and power of CRIQUET are available. The schemas, near the frames, permit a declarative and uniform representation.

PLAN

Introduction

1) Notion d'objet

- Les objets
- Présentation générale des frames
- L'utilisation des frames
- Caractéristiques
- Critiques

2) Les schémas

- Définition et structure
- Classes et instances
- Utilisation des attributs typés
- Les facettes
- Les réflexes
- Une représentation uniforme

3) L'organisation de la base de schémas

- Organisation générale
- Héritage et multi-héritage

4) Cohérence de la base de schémas

- Réflexes
- Spécialisation à l'affectation

5) La dynamique dans la base de schémas

- Définition de la dynamique
- Expression de la dynamique
  - Filtres
  - Schémas traitements
- Les mécanismes d'inférence
  - Filtrage
  - Attachement procédural
  - Valeur par défaut
  - Héritage

6) Manipulation des schémas

Manipulations des classes
defsch, modsch, supsch
Manipulations des instances
defind, supind
Manipulations des attributs
ajatt, afatt, modatt, supatt, obtatt
Manipulations globales
lsch, lind, supbsch, supbind,
charger, sauver, sortir, lire
chind, executer, spec, impression, visu

7) Interactions schémas-règles

Utilisation de règles dans les réflexes

8) Interactions règles-schémas

Langage d'expression
Utilisation de fonctions dans les règles

9) Intérêts des représentations centrées objets

Annexes

Bibliographie

INTRODUCTION

Les systèmes informatiques actuels, afin d'être capables de comportements intelligents, doivent savoir manipuler d'importantes quantités de connaissances et expertises.
Il apparaît que seule une représentation autorisant la description, avec les informations de toutes sortes qui y sont attachées, des objets et traitements utilisés est raisonnablement compatible avec cet objectif.

Deux courants d'idées sont à l'origine de la représentation des connaissances à base d'objets. D'une part, les réflexions de M.Minsky (Minsky 75) sur les "frames", structures de données représentant une situation prototypique, d'autre part le développement de langages et d'environnements de programmation orientée objets tels que Smalltalk (Goldberg 83).

Notre objectif dans ce rapport n'est pas de détailler ces différentes notions, ce qui est fait dans (Bonnet 84, Rechenmann 84, Vignard 85b/c/), mais d'exposer une représentation à base de schémas (Rechenmann 85), proches des "frames", et son intégration dans le système de production CRIQUET (Vignard 84,85a/).

Des utilisations du logiciel CRIQUET par des laboratoires publics et privés dans divers domaines ont permis de mieux définir des outils nécessaires pour réaliser correctement une approche système expert. L'utilisation d'une représentation centrée objets est apparue nécessaire pour des raisons que nous détaillerons par la suite.

Des descriptions plus précises des représentations à base de schémas sont données dans (Rechenmann 85) et du logiciel CRIQUET dans (Vignard 85a/).

I/ NOTIONS D'OBJETS

I-1/ Les objets

Dans un formalisme relationnel un objet n'existe pas en tant que tel mais seulement comme participant à un ensemble d'énoncés dispersés dans la base de connaissances. Le raisonnement consiste à manipuler ces énoncés afin d'en déduire de nouveaux. C'est le cas dans les systèmes de production. Les objets ne sont pas représentés comme des entités à part entière. Ils ne sont décris qu'a travers des propriétés disséminées dans les règles.

A l'opposé, dans le formalisme centré objets, les entités ont une existence réelle et un espace propre dans la base de connaissances. Elles rassemblent la connaissance associée à un objet. Le terme d'objet recouvre plusieurs notions. Il peut désigner un objet au sens physique, mathématique ou au sens du domaine concerné. Des informations descriptives et au sujet de la dynamique de l'objet décrit sont rassemblées.

Les représentations en objets structurés sont nées de la conjonction d'idées de diverses sources. Elle s'inspirent des "schémas" résulats d'études en psychologie par Bartlett, des "frames" de Minsky (Minsky 75), des "scripts" de Schank (Pinson 83), des "objets" utilisés dans les langages orientés objets comme SMALLTALK (Goldbegr 83) ou des types abstraits.

Nous présentons ici uniquement l'utilisation des objets pour le développement d'une représentation des connaissances, sans parler de leur utilisation dans d'autres applications.

I-2/ Présentation générale des frames

L'idée des frames est issue des travaux de Minsky (Minsky 75). Ce dernier est parti de l'idée que nous avons en mémoire des structures stéréotypées d'informations et que nous sélectionnons, chaque fois qu'une situation nouvelle se présente, l'un de ces stéréotypes en tentant de le faire coincider avec les données de la situation courante. Le stéréotype identifié complète notre connaissance et indique alors comment réagir aux événements qui peuvent survenir.

La dynamique consiste à diriger la progression du système des frames devenus inadéquats vers d'autres frames quand la situation évolue.

Le frame (Hayes 81, Kuiper 75) est une généralisation des réseaux sémantiques dans le cas de "noeuds" complexes. C'est une structure de données composée d'un groupe de "slots" (attributs), chacun décrit par un ensemble de "facets" pouvant désigner un frame ou être un simple identificateur.

La structure se schématise ainsi:

```
(frame_nom
      (slot_1   (facet_11 val_11)
                (facet_12 val_12)

                 ...
                (facet_1n val_1n))

      ...

      (slot_p   (facet_p1 val_p1)

                (facet_pq val_pq)))
```

Les entités ainsi décrites sont reliées entre elles (Brachman 83) et forment le plus souvent une arborescence, parfois un réseau. Les liens hiérarchiques sont de type généralisation/spécialisation entre concepts (Goodwin 79).

Ces liens permettent d'utiliser un mécanisme d'héritage par lequel un objet hérite des propriétés des entités qui lui sont hiérarchiquement supérieures.

Des facettes rendent les entités actives à l'aide de réflexes. Ce sont des procédures attachées aux slots, automatiquement activées lors d'un certain type de manipulation sur le slot concerné. Le réflexe "si_besoin" par exemple, désigne la procédure à utiliser pour obtenir la valeur d'un slot si elle manque. Les réflexes "si_ajout" et "si_supprim" indiquent ce qu'il faut faire en cas d'ajout et de suppression de la valeur du slot.

L'attachement procédural est souvent le principal mécanisme d'exploitation de cette représentation.

I-3/ L'utilisation des frames

Dans la plupart des systèmes (Goldstein 80), un frame spécifie la situation initiale avec les informations disponibles, décrivant ainsi le problème à traiter. Le raisonnement consiste à reconnaître dans le monde des objets connus, l'objet le plus précis et le plus proche de la situation donnée. Cet objet permet de compléter la description de la situation initiale et les informations supplémentaires qu'il apporte doivent être les réponses à toutes les questions posées.

Le processus d'identification utilisé est celui de "pattern-matching" ou filtrage, qui permet d'évaluer la proximité de deux frames.

I-4/ Caractéristiques

Parmi les principales caractéristiques de la représentation à l'aide d'objets, on peut citer les suivantes:

-une représentation déclarative avec laquelle le monde est vu comme un ensemble d'objets autonomes, chacun ayant une existence réelle dans la base de connaissances. Un objet est une structure informatique composée d'un ensemble d'attributs. Chaque attribut apporte un élément d'information décrit de façon plus ou moins précise.

-chaque objet comporte à la fois une composante statique et une composante dynamique. On utilise ainsi une puissante combinaison de déclaratif et procédural. A un attribut peuvent être attachées des procédures qui sont appelées à l'occasion d'opérations demandées sur cet attribut.

-les notions de hiérarchie d'objets et d'héritage jouent un rôle essentiel dans l'écriture et la manipulation de la base de connaissances. Un objet transmet sa propre connaissance aux objets qu'il domine.

I-5/ Critiques

De telles structures peuvent être complexes à construire et à manipuler. Le manque d'uniformité dans le langage d'expression contribue à cette complexité apparente.

L'insertion directe du procédural sous forme d'appels de programmes dans les réflexes nuit à l'homogénéité et à la lisibilité de la représentation.

L'aspect déclaratif n'est pas assez développé. Les types de facettes disponibles ne sont pas suffisants pour exprimer de façon déclarative toutes les informations disponibles.

II/ LES SCHEMAS

II-1/ Définition et structure

La structure d'un schéma est conforme aux idées exposées précedemment (Bobrow 77a/). Nous définissons un schéma (Rechenmann 85) comme une liste à trois niveaux d'imbrication. Au premier niveau se situe le nom du schéma, au deuxième la liste de ses attributs. Chaque attribut est décrit au troisième niveau par une liste de facettes, chacune possédant une valeur. Dans le formalisme Lisp, un schéma est représenté par une liste (Winston 81):

```
(nom_schéma
     (attribut 1  (facette 11 valeur 11)
                  (facette 12 valeur 12)
                     ...
                  (facette 1n valeur 1n))
        ...
     (attribut m  (facette m1 valeur m1)
                     ...
                  (facette mp valeur mp)))
```

Les attributs sont choisis par le concepteur et apportent la sémantique de l'objet, c'est à dire sa signification propre, alors que les facettes font partie d'un ensemble prédéfini.

Mais la puissance d'expression à l'aide des schémas est essentiellement liée à la diversité et la signification des différentes facettes utilisables.

On appelle "méta-attributs", des attributs dont le nom est fixé et qui sont interprété de façon particulière par le système:

-est_un : relie une instance à son schéma de classe, lien concept-représentant.
-sorte_de : relie un schéma de classe à un ou plusieurs autres schémas qu'il spécialise.
-spec : relation inverse de sorte_de, relie un schéma de classe à ses spécialisations.
-inst : relie un schéma de classe à ses instances.
-lui_même : a pour valeur, pour toute instance, la référence à elle même et permet ainsi des auto-référence.

II-2/ Deux types de schémas : classe et instance

Un schéma définit une classe, c'est à dire une famille d'objets. Un objet particulier de cette famille est aussi décrit par un schéma, appelé individu ou instance. Un individu ne diffère de la classe dont il est un représentant que par la donnée supplémentaire de

valeurs d'attributs.

Une instance peut être complète ou partielle si tous les attributs sont instanciés ou pas. Une instance hérite directement de son schéma père auquel elle est liée par l'attribut "est_un".

Chaque instance porte un nom qui peut avoir été donné par l'utilisateur ou généré par le système.

Des exemples de schémas de classe et d'instance sont:

La définition d'une personne à l'aide d'un schéma de classe:

```
(personne
    (sorte_de ($valeur être_animé))
    (nom ($un chaine))
    (prenom ($un chaine))
    (age ($un reel)
         ($intervalle 0 120))
    (sexe_m ($un booleen))
    (salaire ($un réel))
    (date_naissance ($un date)))
```

de même pour une date:

```
(date
    (jour ($un entier)
          ($intervalle 1 31))
    (mois ($un chaine)
          ($domaine "janvier" "fevrier" "mars" "avril" "mai"
                    "juin" "juillet" "aout" "septembre" "octobre"
                    "novembre" "decembre"))
    (annee ($un entier)
           ($intervalle 1900 2000)))
```

dont des instances sont:

```
(date1
   (est_un ($valeur date))
   (jour ($valeur 29))
   (mois ($valeur "mai"))
   (annee ($valeur 1951))))
```

```
(francois
    (est_un ($valeur personne))
    (prenom ($valeur "francois"))
    (date_naissance ($valeur date1)))
```

II-3/ Utilisation des attributs typés

Chaque attribut est typé à l'aide des facettes particulières, "$un" ou "$liste_de". On définit ainsi le type des valeurs des attributs. Ce type est:

-soit simple: entier, réel, booleen, chaine de caractères ou s_expr.
-soit complexe, dans ce cas c'est un schéma défini par ailleurs.

Dans ce dernier cas, les valeurs de l'attribut doivent être des instances du schéma donné comme type, ou d'un schéma plus spécifique.

Le type indique ainsi le contenu de l'attribut, sa signification propre c'est à dire sa sémantique.

La distinction entre les types "$un" et "$liste_de" repose sur le nombre de valeurs possibles pour l'attribut en question.

II-4/ Les facettes

Les facettes sont réparties en plusieurs catégories, leur nom toujours préfixé par un "$".

*-les facettes de typage, "$un" et "$liste_de", présentées ci-dessus. Des spécialisations locales sont possibles en donnant un filtre comme valeur à la facette de typage utilisée.

Un exemple est le suivant:

```
(personne_née_en_été
   (sorte_de ($valeur personne))
      . . .

   (date_naissance
       ($un (date
              (mois ($domaine "juin" "juillet"
                              "aout" "septembre"))))))
```

Les instances de "personne_née_en_été" ont une date de naissance dont le mois est juin, juillet, aout ou septembre.

*-les facettes permettant des restrictions de types.
La facette "$intervalle", uniquement utilisable pour des attributs de type élémentaire, définit l'intervalle des valeurs admissibles.

Exemple:

```
(age ($un entier)
     ($intervalle 0 120))
```

La facette "$domaine" définit en extension le domaine des valeurs admissibles.

Exemple:

```
(mois ($un chaine)
      ($domaine "janvier" "février" "mars" "avril" "mai" "juin"
                "juillet" "aout" "septembre" "octobre" "novembre"
                "décembre"))
```

*-la facette "$valeur" est suivie de la (ou des) valeur(s) de l'attribut.

Une instance du schéma classe "personne":

```
(personne_1
    (est_un ($valeur personne))
    (nom ($valeur "durand"))
    (prénom ($valeur "pierre"))
    (age ($valeur 25))
    (sexe_m ($valeur vrai))
    (salaire ($valeur 7000.0))
    (date_naissance ($valeur date2)))
```

Les valeurs sont:

-soit des réels, entiers, booléens ou chaines de caractères comme pour les attributs "nom, prénom, age, sexe_m, salaire".

-soit des références directes à d'autres instances, c'est à dire des noms d'instances, comme dans l'attribut "date_naissance".

Dans les schémas de classes, les valeurs peuvent être des filtres. Un filtre est un schéma, spécialisation d'un schéma existant, qui décrit les instances pouvant être données comme valeur à l'attribut auquel il est attaché.

Un exemple est donné ci-dessous. Un filtre décrit les individus suceptibles d'êtres des amis dans le schéma "personne":

```
(amis ($liste_de personne)
      ($valeur
         (personne
             (age ($intervalle 20 40))
             (nom ($domaine "pascale" "dominique" "francois"
                                                  "jean-luc")))))
```

Lors d'une interrogation sur l'attribut "amis", ce filtre est utilisé pour rechercher toutes les instances convenables, c'est à dire ayant les caractéristiques décrites.

Les filtres peuvent être imbriqués. Un attribut dans un filtre peut avoir pour valeur un autre filtre.

Les facettes "$valeur" peuvent être suivies de plusieurs filtres qui sont essayés séquentiellement. Si l'attribut est typé par "$un", la première instance satisfaisant le filtre, et les restrictions attachées à l'attribut s'il y en a, est retenue. Sinon, tous les filtres sont utilisés de façon exhaustive et toutes les instances convenables sont retenues.

*-la facette "$import" contient un réel entre 0 et 1. Elle indique ainsi l'importance relative de l'attribut dans la description d'ensemble de l'objet. Cette information sera utilisée par la suite pour l'implémentation de processus de filtrage flou.

*-la facette "$défaut" permet d'associer une valeur par défaut à un attribut. Cette valeur n'est prise en compte comme valeur d'attribut que si les autres moyens d'obtention de la valeur ont échoué, au niveau du schéma classe où elle apparaît.

*-la facette "$variable" permet d'associer une variable à un attribut et par la même permet des référemces aux valeurs de cet attribut. Les noms de variables sont préfixés par =, & ou ?.

(nom ($variable =x)) signifie que la valeur de l'attribut "nom" se trouvera dans la variable "x" dès sa détermination.

(nom ($variable &x)) signifie que la valeur de l'attribut "nom" doit se trouver dans la variable "x" lors de manipulations.

(nom ($variable ?x)) signifie que la valeur de l'attribut "nom" résulte d'un filtrage ou d'un attachement procédural.

Exemple:

```
(personne
   (lui_même ($variable =lui_même))
       ; variable pour pouvoir manipuler des auto-références

   (date_naissance
           ($un date)
           ($variable =y))
       ; variable pour pouvoir référencer la date de naissance
       ; ailleurs dans le schéma

   (age  ($un entier)
         ($variable =x)
         ($si_besoin
            (compter_années
                  (origine ($variable &y))
                   ; référence par un "&", donc le système
                   ; recherche une valeur pour "y" ,
                   ; trouvée grâce à "=y" ci-dessus
                  (années ($variable ?x)))))
                   ; la variable "x" est calculée par cet
                   ; attachement. L'instanciation est propagée
   (père_de
          ($un personne)
          ($valeur
             (personne
                (a_pour_père ($variable &lui_même))))))
```

II-5/ Les réflexes

Des facettes permettent l'utilisation des réflexes tels qu'ils ont été définis précédemment.

*-la facette "$si_besoin" réalise l'attachement procédural. Elle est suivie d'un filtre, au sens défini précédemment, auquel est associé un traitement sous forme d'une fonction Lisp. Lors de l'activation du réflexe, ce filtre est instancié dans la mesure du possible. L'instance créée est transmise à la fonction Lisp associée, qui est ainsi activée et qui complète, suivant son code propre, l'instance passée en paramètre. Les valeurs calculées sont propagées par les variables associées.

Plusieurs filtres peuvent suivre la facette "$si_besoin". Ils sont activés séquentiellement jusqu'à obtention d'une valeur acceptable. Un exemple est donné dans le schéma classe "personne" donné ci-dessus. Au schéma décrivant le traitement "compter_années":

```
(compter_années
      (origine ($un date))
      (années ($un entier)))
```

est associé une fonction Lisp de même nom:

```
(de compter_annees (inst)
    (aj_valeur inst
       'annees
       (-  (car (val_facette inst 'date_cour '$valeur))
           (car (val_facette
                    (ref (car (val_facette inst 'origine '$valeur)))
                    'annee
                    '$valeur)))))
```

La facette "$si_besoin_choisir" permet de faire afficher à l'écran les méthodes disponibles décrites par les filtres qui suivent la facette. Le choix est alors laissé à l'utilisateur.

Un exemple est le suivant, décrivant l'attribut "age":

```
(age  ($un entier)
      ($variable =x)
      ($si_besoin_choisir
         (compter_années
               (origine ($variable &y))
               (années ($variable ?x)))

         (interroger
               (réponse ($variable ?x)))))
```

*- les facettes réflexes "$si_ajout", "$si_modif", "$si_supprim" sont aussi suivies de la description d'une procédure, appelée respectivement en cas d'ajout, de modification ou de suppression d'une valeur.

*-la facette "$a_vérifier" peut être également suivie de plusieurs filtres. Les fonctions attachées sont dans ce cas des prédicats, c'est à dire des fonctions qui renvoient un résultat booléen. Ces prédicats sont tous évalués dès qu'une valeur doit être attribuée à l'attribut. Ils doivent tous être vérifiés pour que la valeur soit admissible.

En voici un exemple:

```
(âge ($un entier)
     ($variable =y)
     ($a_vérifier
         (positif (nb ($variable &y)))))
```

La syntaxe des filtres est la même dans tous les cas, seule l'interprétation change.

*- des facettes de contrôle "$si_échec" et "$si_succès" indiquent les actions à entreprendre en cas respectivement d'échec ou de succès d'obtention de la valeur de l'attribut auquel elles sont attachées.

*-des facettes décrites par la suite facilitent le passage de la représentation externe des valeurs à la représentation interne (lecture) et inversement (impression).

II-6/ Une représentation uniforme

Par définition, pour toute facette, toute valeur est un filtre, c'est à dire un descripteur plus ou moins précis.

La valeur des facettes "$valeur" peut indiquer à l'aide d'un filtre les caractéristiques que doit satisfaire un individu pour être une valeur admissible. La facette "$si_besoin" indique le traitement à utiliser à l'aide d'un filtre appelé "schéma traitement". Ces derniers sont structurellement identiques aux autres. Comme leur nom l'indique, ils décrivent le ou les traitements utilisables pour obtenir la valeur cherchée. Certains attributs décrivent les conditions d'application du traitement et les paramètres nécessaires, d'autres correspondent aux résultats. Chaque schéma traitement est une description facilement accessible et modifiable d'une procédure codée par ailleurs et qui sera activée lors d'une référence au schéma traitement associé.

En se satisfaisant de cette expression de la dynamique propre aux schémas, le recours à d'autres représentations et à leurs mécanismes d'exploitation est évité. Il en résulte une unité du formalisme, une simplicité et une bonne lisibilité des expressions.

De plus, le système a une connaissance complète des traitements disponibles, décrits en particulier avec leurs conditions d'application. Il peut donc choisir le schéma traitement adéquat.

Par exemple, en considérant des traitement d'intégration de systèmes différentiels, on peut définir les deux schémas suivants:

```
(Runge_Kutta_4
     (sorte_de ($valeur intégration))
     (système ($un système_classique))
     (solution ($un solution)))


(Gear
   (sorte_de ($valeur intégration))
   (système ($un système_raide))
   (solution ($un solution)))
```

associés à des fonctions Lisp de même nom. Le type de l'attribut "système" différencie ces deux traitements et constitue la condition de choix entre eux deux.

Dans les frames classiques, avec la dynamique spécifiée uniquement dans les réflexes sous forme d'un appel direct de programmes, le système ne pourrait qu'indiquer le nom du programme utilisé sans donner plus d'informations sur la signification du traitement et les raisons de son utilisation.

III/ L'ORGANISATION DE LA BASE DE SCHEMAS

III-1/ Organisation générale

Tout schéma s'inscrit dans une structure de demi treillis où il domine des schémas plus spécifiques et est dominé par des schémas plus généraux. Ces liens sont décrits dans les méta-attributs "sorte_de" et "spec". Ce sont des "liens verticaux".

Un schéma de classe peut être spécialisé par enrichissement, c'est à dire par addition de nouveaux attributs ou par spécialisation des attributs existants. La spécialisation d'un attribut peut se faire en rajoutant de nouvelles facettes ou en restreignant les valeurs de certaines facettes déjà affectées. On peut préciser un type plus spécifique ($un ou $liste_de), restreindre le domaine des valeurs admissiles ($intervalle ou $domaine) ou les conditions associées à l'attribut ($a_vérifier) ou enfin donner dans "$valeur" un filtre plus précis que celui existant.

Les individus sont des noeuds terminaux de la hiérarchie. La spécialisation décrite ci-dessus interdit la redéfinition d'un attribut. En particulier, il n'est pas possible d'affecter à un attribut d'une instance une valeur autre que celle qui aurait été donnée au niveau du schéma de classe.

A côté des relations verticales, des "liens horizontaux" sont aussi utilisés, sous forme de références imbriquées. Les facettes "$un" et "$liste_de" permettent de spécifier respectivement des liaisons (1:1) et (1:n).

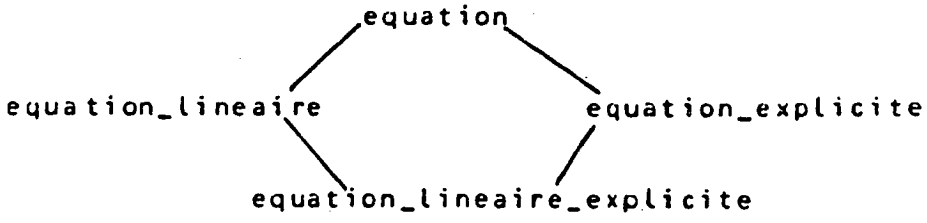
III-2/ Héritage et multi-héritage

Un des intérêts de l'organisation des schémas tient à l'utilisation de mécanismes d'héritage par lesquels un schéma recoit la connaissance associée à ses ancêtres.

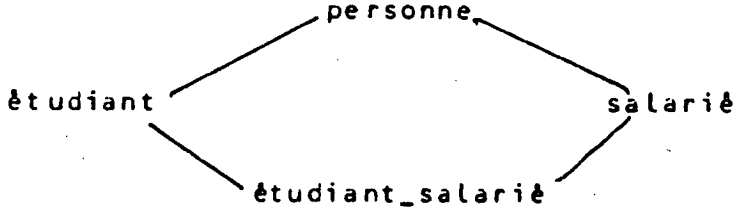
Ce mécanisme facilite la définition de nouveaux schémas en évitant la recopie d'informations redondantes et permet une mise à jour plus sûre et rapide. Les modifications apportées à un schéma sont propagées par héritage, ce qui assure une certaine cohérence de la base d'objets.

La structure de treillis implique qu'un schéma puisse hériter de plusieurs autres schémas.

Par exemple, avec différents types d'équations, on peut avoir l'organisation suivante :



ou encore:



Un problème d'héritage est à traiter en cas de conflit entre attributs hérités. Deux attributs sont en conflit s'ils ont le même nom.

La règle de priorité utilisée de façon classique entraine un héritage de bas en haut d'abord puis de gauche à droite. Dans le second exemple donné ci-dessus, l'héritage se fait à partir des entités suivantes, dans l'ordre donné : étudiant_salarié, étudiant, personne et salarié.

De façon plus précise, le multi-héritage offert consiste à faire la fusion des descriptions des attributs en conflit, en respectant la priorité d'héritage citée.

L'héritage multiple accroît encore les possibilités de spécification des connaissances, en permettant par exemple des points de vue multiples d'un même concept. Le concept "Equation_lineaire_explicite" correspond à la combinaison de deux points de vue différents d'une même équation.

L'organisation reste dynamique. Les liens entre schémas ne sont résolus que lors des interrogations, aucune référence n'est figée. Cet aspect facilite la propagation, qui en fait n'a pas lieu d'être. La restructuration de la base est possible, en modifiant la liste des ancêtres des schémas utilisés.

Une telle organisation dynamique est certe plus coûteuse en temps mais facilite la spécification des connaissances et reste en accord avec l'aspect évolutif et dynamique de la mémoire chez l'être humain.

IV/ COHERENCE DANS LA BASE DE SCHEMAS

IV-1/ Les réflexes

Le maintien de la cohérence de l'ensemble de la base, c'est à dire des relations entre schémas, est assuré par l'emploi de réflexes avec les facettes "si_ajout", "si_supprim" et "si_modif" qui décrivent les actions à éxecuter lors d'ajout, de suppression et de modification d'une valeur d'attribut.

Les facettes "$domaine", "$intervalle" et "$a_vérifier" permettent aussi de spécifier des contraintes d'intégrité qui doivent être vérifiées lors de l'instanciation des attributs.

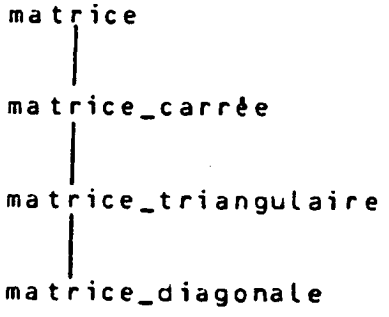
IV-2/ Spécialisation à l'affectation

Lors de l'affectation de la facette "$valeur" d'un attribut, le système vérifie la cohérence de l'individu donné en valeur par rapport au type de l'attribut. Il vérifie que cet individu est bien le représentant d'une classe hiérarchiquement inférieur ou égal au schéma qui est donné comme type de l'attribut instancié. Si ce n'est pas le cas, il tente de spécialiser cet individu autant qu'il le faut pour permettre l'affectation.

Par exemple, avec le schéma:

```
(inversion
    (matrice  ($un matrice_carrée))
    (ordre  ($un entier))
    (inverse ($un matrice)))
```

si la valeur donnée pour instancier l'attribut "matrice" est une instance du schéma "matrice" et non pas de "matrice_carrée", le système essaie de spécialiser cet individu et de le faire descendre dans la hiérarchie:



Pour cela, le système regarde si l'individu manipulé comme une "matrice" vérifie les contraintes supplémentaires utilisées pour caractériser une "matrice_carrée".

V/ LA DYNAMIQUE DANS LA BASE DE SCHEMAS

V-1/ Définition de la dynamique

Utiliser un schéma consiste essentiellement à:

-créer des schémas de classe qui s'inscrivent dans une structure de treillis induite par les relations de généralisation/spécialisation décrites à travers les méta-attributs "spec" et "sorte_de".

-créer des instances complètes ou partielles de ces schémas de classe.

-chercher des valeurs pour des attributs non affectés à la création des instances. Cette opération d'instanciation est le mécanisme fondamental d'inférence dans une représentation centrée objets.

V-2/ Expression de la dynamique

Le système recherche des valeurs d'attributs à l'aide des informations apportées dans les facettes "$valeur" et "$si_besoin".

*-La facette "$valeur" contient:

-soit des valeurs simples ou des noms des instances constituant la valeur.

-soit un ou plusieurs filtres qui donnent les caractéristiques des individus pouvant instancier l'attribut.

L'exemple de filtre donné ci-dessous caractérise les petits fils d'une personne et permet ainsi la recherche, parmi toutes les instances de "personne", des petits fils d'un individu donné.

```
(personne
   (lui_même ($variable =lui_même))
   (a_pour_père ($un personne))
   (grand_père_de
         ($liste_de personne)
         ($valeur
            (personne
               (a_pour_père ($valeur
                              (personne
                                (a_pour_père
                                   ($variable &lui_même)))))))))
```

*-La facette "$si_besoin" contient un ou plusieurs schémas traitements, tels qu'ils ont été définis précédemment. Ce sont aussi des filtres, décrivant des traitements activables pour calculer une

valeur d'attribut. Des exemples ont été donnés précédemment.

V-3/ Différentes inférences

En toute occasion, pour déterminer une valeur d'un attribut, le système exploite successivement les facettes, dans l'ordre donné:

-$valeur
-$si_besoin
-$défaut

Les différentes techniques d'inférence sont donc:

a)Le premier cas correspond à:

-soit une inférence directe si "$valeur" contient une valeur simple, ou la description exacte d'une instance (le nom).

-soit une inférence par filtrage si "$valeur" contient un ou plusieurs filtres. Le système résoud ces filtres, c'est à dire qu'il recherche par filtrage, ou pattern-matching, les instances les satisfaisant.

b) Le second cas correspond à une inférence procédurale. Les valeurs résultent de l'activation du traitement algorithmique décrit par le schéma traitement donné derrière la facette "$si_besoin". Certains attributs de ce schéma correspondent aux paramètres d'entrée et de sortie de l'algorithme. Une instance de chaque schéma traitement est créée lors de chaque exécution effective du traitement.
La première étape d'une inférence procédurale consiste à filtrer l'ensemble des instances existantes du schéma traitement utilisé. Si aucune instance adéquate n'est trouvée, le traitement algorithmique est exécuté, calculant les paramètres de sortie. Si leurs valeurs sont admissibles, une nouvelle instance est créée. Par effet de bord, grâce aux variables dans "$variable", l'attribut auquel est attaché le traitement par la facette "$si_besoin" peut recevoir sa valeur.
Un exemple est:

```
(matrice
  (lui_même ($variable =lui_même))
  (ordre ($un entier)
         ($variable =x))
  (element ($liste_de reel))
  (inverse ($un matrice)
           ($variable =y)
           ($si_besoin
              (gauss
                 (matrice ($variable &lui_même))
                 (ordre ($variable &x))
                 (inverse ($variable ?y))))))
```

Le schéma traitement décrit la méthode d'inversion de Gauss. Quand on recherche l'inverse d'une matrice donnée:

-le système regarde si cet objet n'a pas déjà été calculé, c'est à dire s'il n'existe pas une instance du schéma traitement Gauss correspondant a ce calcul.

-si ce n'est pas le cas, l'algorithme est activé et permet d'instancier l'attribut "inverse" dans la nouvelle instance du schéma "Gauss". La valeur de la variabble "y" est propagée pour ainsi instancier l'attribut voulu, qui est l'attribut "inverse" dans le schéma "matrice".

Un module d'interface traitement assure la mise en oeuvre et le contrôle de l'exécution des programmes utilisateur décrits par des schémas traitements. Des outils techniques peuvent être implémentés pour permettre l'interface avec des programmes écrits dans un autre langage que Lisp (Bensaid 84).

c) Le dernier cas est une inférence par défaut. Si les technqiues précédentes ont échoué, on utilise la valeur par défaut s'il y en a une.

d) L'inférence par héritage est utilisée en cas d'échec de toutes les techniques citées ci-dessus.
Il y a alors, au niveau du schéma courant, activation des actions décrites derrière la facette "$si_échec", si elle existe. Puis le système exploite les même facettes, dans le même ordre, mais dans les schémas de classe qui domine le schéma courant.
L'héritage se fait selon le type parcours cité précédemment, de haut en bas d'abord, puis de gauche à droite.

En cas de succès d'obtention de la valeur de l'attribut, les actions spécifiées derrière la facette "$si_succès", si elle existe, sont activées.

VI/ MANIPULATION DES SCHEMAS

Après avoir décrit les schémas et leur dynamique propre, nous présentons par la suite les commandes de manipulation des schémas et les outils disponibles pour permettre des interactions entre les règles et les schémas.

Dans la plupart des commandes citées, les paramètres sont réclamés par le système s'ils ne sont pas donnés à l'appel, ou possèdent des valeurs par défaut.

VI-1/ Manipulation des schémas de classe

DEFSCH

Permet de définir un nouveau schéma.
Pour la saisie du texte, on utilise l'éditeur pleine page "pépé", dont toutes les commandes sont utilisables. Une fois le texte du schéma construit, la commande "CTRL W" permet de vérifier qu'il constitue une description correcte. Si ce n'est pas le cas, l'utilisateur peut abandonner la définition ou corriger les erreurs signalées.

MODSCH nom-schéma

Permet de modifier le texte d'un schéma.
L'éditeur pleine page "pépé" est utilisé de la même manière qu'avec "defsch". La clé "CTRL W" doit aussi être utilisée pour faire enregistrer le nouveau texte du schéma.
Peu de vérifications sont faites après les modifications, qui doivent donc être faites avec prudence pour conserver la cohérence de la base de schémas.

Lors de la création et de la modification de schémas, deux types essentiels de vérifications sont pratiquées:
-les premières permettent de s'assurer de la validité de la syntaxe du texte.
-les secondes vérifient en partie la validité du contenu du schéma par rapport aux schémas qui lui sont hiérarchiquement supérieurs. Le méta attribut "sorte_de" doit être précisé. Un schéma est correct si tous ses attributs le sont. Tout attribut doit posséder un type. Le système s'assure de la validité de la valeur donnée par rapport à ce type et aux contraintes s'il y en a ($intervalle, $domaine, $a_vérifier). Un attribut redéfini doit être spécialisé. Les facettes "$valeur, $intervalle, $domaine" doivent contenir des valeurs plus spécifiques que celles données dans les schémas supérieurs.

SUPSCH nom-schéma

Permet de supprimer un schéma de classe.
Les instances et tous les schémas de classe hiérarchiquement inférieurs sont aussi supprimés, et cela de façon récursive. Dans les schémas dominants qui subsistent, le méta-attribut "spec" est mis à jour.

VI-2/ Manipulation des instances

DEFIND nom-schéma nom-instance

permet de définir une instance, en interactif.
Si le nom d'instance n'est pas donné, le système génère un nom par défaut. Le système réclame une valeur pour chaque attribut.

Si un attribut doit avoir pour valeur un individu alors le système peut créer une instance du type voulu et continuer le processus d'instanciation pour cette nouvelle instance. Ce principe s'applique de façon récursive tant que l'utilisateur le désire et que les attributs ont pour type un schéma ayant une existence réelle dans la base.

Pour faciliter la définition, l'utilisateur a le choix entre diverses réponses aux interrogations du système:

- "~" pour ne pas fournir de valeur ou terminer une liste

- "?" pour avoir le type de la valeur attendue.

- "<" pour en finir avec l'instanciation en cours. Les attributs restant reçoivent la valeur manquante.

- ">" qui ne peut être employé que si la valeur attendue est de type complexe. Dans ce cas:

  -soit on donne le nom d'une instance existante.

  -soit on répond ">" et le système créé une nouvelle instance du type voulu, dans laquelle il se déplace pour continuer la définition.

Si un attribut est du type simple chaine, la valeur à fournir doit être donnée entre guillemets. Les noms d'instances par contre ne doivent pas être quotés.

SUPIND nom-instance

Permet de supprimer une instance.
Le système met à jour le méta-attribut "inst" dans le schéma de classe père de cette instance. Mais le système ne gère pas toutes les références à cette instance, en particulier par des liens horizontaux. Il y a donc, si on ne prend aucune précaution, des risques d'introduire des références non résolues.

VI-3/ Manipulation des attributs

Nous présentons les fonctions d'accès et de manipulation des valeurs des attributs (facette "$valeur").

Si un paramètre manque à l'appel de la commande, le système redemande tous les paramètres nécessaires en interactif.

Les réflexes adéquates sont activés selon la commande utilisée:

Commandes	Réflexes
afatt, ajatt	$si_ajout
obtatt	$si_besoin
supatt	$si_supprim
modatt	$si_modif

Les commandes disponibles sont les suivantes:

AJATT nom-schéma nom-attribut valeur

Pour ajouter une valeur à un attribut choisi.

AFATT nom-schéma nom-attribut valeur

Pour affecter une valeur à un attribut choisi. La valeur donnée écrase toute valeur précédente s'il y en avait une.

MODATT nom-schéma nom-attribut valeur1 valeur2

Remplace "valeur1" par "valeur2" dans la valeur de l'attribut choisi.

SUPATT nom-schêma nom-attribut valeur

Supprime la valeur donnée dans l'attribut choisi.

OBTATT nom-schêma nom-attribut

Recherche la valeur de l'attribut donné.

VI-4/ Manipulations globales

Les commandes suivantes concernent pour la plupart les bases de schêmas et d'instances dans leur ensemble.

LSCH

Pour imprimer la liste des noms de schêmas de classe existants.

LIND

Pour imprimer la liste des noms d'instances existantes.

SUPBSCH

Pour supprimer tous les schêmas de classe existants.

SUPBIND

Pour supprimer toutes les instances existantes.

SAUVER nom-fichier type

Permet de sauvegarder dans un fichier de nom donné la description des schêmas de classe et des instances existants au moment de la sauvegarde.

Le type donné doit être "bo", pour indiquer au système la sauvegarde des bases d'objets. La même commande est utilisable pour sauvegarder des bases de règles, de méta-règles ou autres (Vignard 85d/).

Chaque schêma est sauvegardé sous une forme lisible, paragraphée. Cette mise en forme est faite avec un processus spécial d'impression. Le format à respecter est détaillé à la commande suivante.

CHARGER nom-fichier type

Permet de charger les définitions de schémas contenues dans le fichier de nom donné. Le type là aussi doit valoir "bo".

Les fichiers utilisés sont:

-soit des fichiers créés avec la commande précédente "sauver", auquel cas il n'y a pas de problèmes au chargement.

-soit des fichiers créés et remplis par l'utilisateur lui-même. Dans ce cas, l'utilisateur doit prendre soin de respecter une certaine syntaxe et grammaire afin que le système puisse enregistrer les définitions des schémas. Il faut:

-donner les défintions de schémas entre "( )".
-donner les définitions de filtres internes entre "{ }".
-terminer le nom de chaque attribut par : "."
-préfixer chaque facette par "$".
-terminer la description de la dernière facette de chaque attribut par ";".
-les valeurs de type simple sont entre guillemets ("xxx").
-les facettes utilisables sont:

$val $valeur	pour $valeur
$var $variable	$variable
$domaine	$domaine
$defaut	$defaut
$inter $intervalle	$intervalle
$un $est-un $est_un	$un
$liste $liste-de $liste_de	$liste_de
$import $importance	$import
$comment $commentaire	$comment
$si-besoin $si_besoin	$si_besoin
$si-besoin-choisir	$si_besoin_choisir
$si_besoin_choisir	
$si-supp $si_supp $si_supprim $si-supprim	$si_supprim
$si_ajout $si-ajout	$si_ajout
$si-modif $si_modif	$si_modif
$regles	$regles
$si-succes $si_succes	$si_succes
$si-echec $si_echec	$si_echec

Il y a de même les facettes de lecture et d'impression citées par la suite.

Un exemple est le suivant:

```
(personne
   spec:
        $valeur personne_1 personne_2  ;
   lui_meme:
        $variable =lui_meme ;
   nom:
        $un chaine ;
   age:
        $un entier
        $intervalle 0 120
        $variable =age
        $si_besoin
           (compter_annees
             origine:
                  $variable &date ;
             annees:
                  $variable ?age ;)
             (interroger
             question:
                  $valeur "age de la personne" ;
             reponse:
                  $variable ?age ;)
             ;
   date_naiss:
        $un date
        $variable =date ;
   a_pour_pere:
        $un personne ;
   pere_de:
        $liste_de personne
        $valeur
          (personne
             a_pour_pere:
                  $variable &lui_meme ;)
             ;
   grand_pere_de:
        $liste_de personne
        $valeur
           (personne
             a_pour_pere:
                  $valeur
                    (personne
                       a_pour_pere:
                            $variable &lui_meme ;)
                       ;)
             ;)
```

Lors du chargement, chaque définition de schéma est traitée avec le même processus que celui utilisé pour la définition des schémas (commande "defsch"). Si des erreurs sont décelées dans la définition des schémas, elles sont signalées. Le schéma en cause

n'est pas enregistré mais le chargement se poursuit. L'ordre des définitions dans le fichier chargé est important. Pour qu'une définition soit acceptée il faut que tous les schémas auxquels elle fait référence aient été définis auparavant.

SORTIR nom-schéma niveau nom-fichier

Permet d'écrire sur le fichier de nom donné l'ensemble des instances directes du schéma donné. Le paramètre "niveau" est un entier utilisé comme dans la commande "visu" décrite par la suite.

LIRE nom-schéma nom-fichier

Permet de lire et de créer des instances du schéma donné à partir du fichier de nom donné.
Les facettes "$avant_lec, $après_lec, $en_lec, $pour_lec et $err_lec" permettent si elles sont utilisées de lire les valeurs d'attributs, et donc les instances, selon un format déterminé.

-$avant_lec et $après_lec sont activées respectivement avant et après la lecture de la ou des valeurs de l'attribut.

-$en_lec est activée lors de la lecture d'une liste de valeurs, après la lecture de chaque valeur de la liste.

-$pour_lec est utilisée pour lire la ou les valeurs proprement dîtes.

-$err_lec est activée en cas d'erreur à la lecture d'une valeur d'attribut.

CHIND filtre

Permet de résoudre un filtre.
Le filtre utilisé est un schéma construit au préalable avec la commande "defsch". Le méta-attribut "sorte_de" dans le filtre indique sa nature et ainsi l'ensemble des instances à utiliser pour le résoudre. La commande rend la première instance satisfaisant le filtre.

EXECUTER nom-schéma

Permet de lancer l'exécution du schéma traitement de nom donné, après instanciation partielle en interactif.

SPEC nom-instance nom-schéma

Permet de spécialiser une instance jusqu'au schéma de classe de nom donné.
Le système tente de rattacher l'instance donnée au schéma de classe spécifié hiérarchiquement inférieur, donc plus spécialisé, que celui auquel elle est attachée initialement. Ce mécanisme est décrit en V-2/.

IMPRESSION nom-schéma niveau

Permet d'imprimer le texte du schéma de nom donné, classe ou instance, à différents niveaux de précision. Le paramètre "niveau" est un entier entre 0 et 3.
Selon sa valeur, plus ou moins d'informations sont présentées:

0 : que les facettes $valeur et de typage;
1 : toutes les facettes mais uniquement les valeurs simples;
2 : toutes les facettes avec les filtres donnés en valeur s'il y en a;
3 : même chose que 2 mais avec résolution des références imbriquées. Les instances valeurs de facettes sont imprimées au niveau 0.

VISU nom-instance niveau

Permet d'imprimer le texte d'une instance de nom donné en utilisant les facettes d'impression "$avant_impr, $après_impr, $pour_impr et $en_impr".

-$avant_impr et $après_impr sont activées respectivement avant et après l'impression de la ou des valeurs de l'attribut.
-$en_impr est activée lors de l'impression d'une liste de valeurs, après l'impression de chaque valeur de la liste.
-$pour_impr est appelée pour imprimer la ou les valeurs proprement dites.

Le niveau est un entier qui indique le nombre de niveaux d'imbrications souhaités.

VII/ INTERACTIONS SCHEMAS-REGLES

Des règles peuvent être données comme valeur dans certaines facettes. Dans ce cas, la syntaxe de la valeur donnée doit être:

valeur = (LR, SP)

SP est un entier entre 0 et 5 indiquant une structure de contrôle à utiliser lors de l'exploitation des règles (Vignard 85d/).

LR est:
- -soit la liste vide "()", auquel cas la base de règles courante est utilisée dans son ensemble.
- -soit une liste de numéros des règles à utiliser dans la base de règles courante.
- -soit le nom d'une base précédemment sauvegardée (Vignard 85d/).

Ces paramètres sont facultatifs. Par défaut le système utilise "(), 0)".

Les règles peuvent être utilisées derrière les facettes:

- -$si_besoin en chaînage arrière;
- -$règles en chaînage avant.

En tenant compte de la nouvelle facette, "$règles", le système utilise pour la recherche d'une valeur, dans l'ordre:

- -$valeur
- -$si_besoin
- -$règles
- -$défaut
- -puis l'inférence par héritage.

VIII/ INTERACTIONS REGLES-SCHEMAS

VIII-1/ Langage d'expression

Le langage d'expression des connaissances utilisé dans CRIQUET (Vignard 85d/) a été modifié pour permettre l'utilisation de filtres et ainsi faire référence aux schémas.

La formule utilisable pour cela, comme prémisse d'une règle est:

```
Il-existe {nombre} {type} {variable} tel que
          {filtre}
```

Le mot clé utilisé par le système pour reconnaître l'utilisation d'un filtre est "il-existe". Le "{filtre}" est un filtre comme il peut en apparaître derrière la facette "$valeur" d'un attribut. Le "{type}" est le nom de la classe concernée par le filtre qui correspond à la valeur du méta attribut "sorte_de" devant apparaître dans le filtre donné. Les recherches, pour résoudre le filtre, se font parmi les instances du concept donné dans le méta attribut "sorte_de".

Selon le {nombre}, "$un" ou "$tous", la variable donnée dans {variable} recoit la première ou toutes les instances vérifiant le filtre. La variable ainsi instanciée peut être manipulée dans le texte de la règle comme telle. Mais il faut tenir compte de la structure de son instanciation. Une notation pointée facilite ces manipulations. Pour parler de la valeur de l'attribut "nom" du schéma "personne" on écrit: "personne.nom". De même, si une instance est contenue dans la variable "?x", on écrit: "?x.nom".

Un exemple de saisie est:

```
(système)       Donnez une condition
(utilisateur)   il-existe $un personne ?x tel que
(système)       Donnez le filtre
(utilisateur)   (filtre1
                    sorte_de:
                         $valeur personne;
                    nom:
                         $valeur "roger";)
(système)       Donnez une condition
(utilisateur)   predicat : ?x.age > 25
(système)       Donnez une condition
(utilisateur)   -
```

La notation pointée n'est pas reconnue dans les règles lors d'un chaînage arrière.

VIII-2/ Utilisation de fonctions dans les règles

Des appels de fonctions sont directement utilisables dans le texte des règles (Vignard 85a/), avec la syntaxe:

fonction : nom-fonction paramètres

Des fonctions ont été créées pour permettre la manipulation des objets selon cette technique. Elles ont la même définition fonctionnelle que les commandes auxquelles elles correspondent, mais nécessitent en paramètre toutes les données qui cette fois ne pourront pas être demandées en interactif.

Fonctions	Commandes correspondantes
*obtatt sch att	obtatt
*ajatt sch att val	ajatt
*afatt sch att val	afatt
*modatt sch att val1 val2	modatt
*supatt sch att val	supatt
*defind sch texte	defind

D'autres commandes sont directement utilisables comme fonctions, telles que "supind, chind, spec, impression". Il est facile de définir et d'enregistrer de nouvelles fonctions (Vignard 85d/).

L'exemple donné ci-dessous montre:

- -l'utilisation d'un filtre dans une règle
- -la manipulation d'une variable instanciée par un objet
- -l'utilisation de fonctions internes

```
( inference 1
  CONDITIONS
   ($un ?x
      (personne
           (sorte_de
              ($valeur personne))
           (nom
              ($valeur "roger")))

    )
   (sachant ?y =  ?x.nom )
   (?z aimer ?y)
   (non (?z etre marie))
  ACTIONS
   (ajoutfait (?z epouser ?y))
   (fonction (*afatt '?x 'epoux_de '?z)
             (impression '?x 1))
  COMMENTAIRES
  ()
  CONSEILS
  ()
  COEFF
  1 )
```

Le langage d'expression, avec la possibilité d'utiliser des fonctions dans le corps d'une règle, permet de simuler une formule telle que:

Pour tout {variable} telle que {filtre} faire {action}

On écrit avec les outils disponibles:

SI il-existe $tous {type} {variable} tel que {filtre}
ALORS fonction : répéter-action {action} {variable}

où la fonction "répéter-action" exécute l'action initiale {action} sur chaque objet contenu dans {variable}.

Les schémas sont utilisables de la même manière dans les méta règles.

IX/ INTERETS DES REPRESENTATIONS CENTREES OBJETS

Des utilisations de CRIQUET dans divers domaines ont révélé des lourdeurs et impossibilités quant à l'expression des connaissances.

Une application au centre INRIA de Sophia Antipolis avait pour objectif une approche système expert dans le domaine de la classification automatique des galaxies (Granger 84). Le problème de classification nécessite dans ce cas la manipulation d'une hiérarchie prédéfinie de prototypes. Le système doit ensuite identifier le prototype le plus proche de l'objet décrivant la galaxie à reconnaître, caractérisée par des informations qualitatives et quantitatives. Après divers essais avec CRIQUET, un système a été réalisé (Thonnat 85) à l'aide d'une représentation centrée objets et de règles de production utilisées pour permettre l'évolution dans une hiérarchie statique d'objets.

Pour une approche système expert dans le domaine de l'édition de la parole synthétisée (chez Texas Instrument), il est apparu nécessaire de pouvoir modéliser et se servir d'un grand nombre de traitements algorithmiques codés en Fortran. Ceci est difficile uniquement à l'aide de règles.

Enfin, dans le domaine de l'étude des bâtiments (au CSTB), il fallait spécifier des connaissances parfaitement structurées. Le manque d'organisation inhérent aux règles de production a entrainé des lourdeurs dans l'expression des connaissances (Halleux 85)

De façon générale, les représentations à base d'objets ont les avantages des représentations déclaratives et procédurales. Elles ont la flexibilité, lisibilité, modularité et modifiabilité des premières. L'organisation des connaissances avec héritage permet une économie de description. La représentation procédurale permet l'expression d'heuristiques souvent impossible autrement.

Organisation des connaissances

La représentation centrée objets est nécessaire quand les connaissances sont structurables. L'organisation avec héritage et multi-héritage facilite la spécification des connaissances et paraît dans bien des cas plus naturelle qu'une mise à plat sous forme d'une simple base de faits. Des processus de propagation le long de la hiérarchie utilisée permettent en partie de maintenir la cohérence de la base d'objets.

Efficacité

Dans les systèmes de production, l'atomisation forcée des connaissances et l'élimination des dépendances entre règles au nom de la modularité supprime toute organisation. Cette organisation, utilisée avec une représentation centrée objets, améliore de beaucoup les performances du système. Avec les notions de contextes, les recherches se font dans des espaces peu importants. Ainsi pour résoudre un filtre, le système ne considère que les instances d'une même famille. Dans un système de production classique, la recherche à chaque cycle des règles activables oblige théoriquement à parcourir toute la base de règles.

Manipulation d'entités réelles

Dans certains domaines, tels que la CAO ou la modélisation de systèmes dynamiques, il est nécessaire de manipuler de véritables objets informatiques. Ceci est impossible à l'aide de simples faits élémentaires comme dans un système de production.

Modélisation de traitements algorithmiques

Au contraire d'autres, la représentation à base d'objets permet aisément la modélisation de traitements algorithmiques. Une telle représentation apporte plus de connaissances sur les traitements que ce qui est permis habituellement, et confère ainsi plus de puissance au système. Les objets constituent un moyen uniforme d'interface entre des environnements de manipulations symboliques et de calcul scientifique.

Self raisonnement, apprentissage

Cette représentation permet de rassembler des informations de diverses natures (qualitatives, quantitatives et sémantiques) sur une même entité. Ces informations, sous forme déclarative, sont accessibles au système lui-même. Il peut donc raisonner sur sa propre connaissance, la modifier et l'améliorer. Des processus d'apprentissage sont envisageables.

Différents points de vues

Les possibilités de multi-héritage, et la flexibilité de la représentation, permettent de considérer un même objet sous différents points de vue.

Raisonnement par défaut, traitement des exceptions

L'utilisation des valeurs par défaut améliore la puissance des raisonnements qui ne sont plus bloqués par l'absence d'une valeur effective.
Les valeurs par défaut permettent aussi de spécifier et manipuler les exceptions. Une exception peut s'écrire:

En général l'attribut A de l'objet O a pour valeur X
Exception: si l'attribut B a pour valeur Y
alors l'attribut A vaut Z

Ce que l'on peut noter par:

```
(O
   A:
     $défaut X;)

(O'
  sorte_de:
     $valeur O;
  A:
     $valeur Z;
  B:
     $valeur Y;)
```

Les objets constituent donc une représentation uniforme pour coder de façon déclarative, accessible et modifiable différents types de connaissances. L'organisation facilite la spécification et améliore la puissance et l'efficacité des raisonnements.

LISTE DE COMMANDES DE MANIPULATIONS DES OBJETS

Defsch : définition de schémas de classe
Modsch : modification du texte de schémas de classe
Supsch : suppression de schémas de classe
Defind : définition en interactif d'instances
Supind : suppression d'instances
Obtatt : obtenir une valeur de facette d'un attribut donné
Ajatt : ajout d'une valeur
Afatt : affectation d'une valeur
Supatt : suppresion d'une valeur
Modatt : modification d'une valeur par remplacement par une autre
Lsch : obtenir les noms de schémas de classe
Lind : obtenir les noms d'instances
Supbsch : suppression de tous les schémas de classe
Supbind : suppression de toutes les instances
Charger : charger un fichier de schémas
Sauver : sauvegarde des schémas existants
Sortir : sauvegardes de certaines instances
Lire : lecture et création d'instances
Chind : résoudre un filtre donné
Spec : spécialiser une instance donnée
Executer : activer un schéma traitement
Impression : imprimer le texte d'un schéma donné
Visu : imprimer une instance

LISTES DES COMMANDES

liste des commandes disponibles

Commandes de manipulations des faits :
ajf : ajouter des faits en conversationnel dans la base de faits
spf : supprimer un fait dans la base de faits
mdf : modifier le coeff de plausibilite d un fait
cbf : construire la base de faits
spbf : detruire la base de faits
bf : imprimer la base de faits
tribf : pour trier la base de faits sur le coeff de vraisemblance
rechbf : recherche associative dans la base de faits

Commandes de manipulations des hypotheses :
bh : imprimer l ensemble hypotheses
ajh : ajouter une hypothese
sph : supprimer une hypothese
spbh : supprimer la base d'hypotheses

Commandes de manipulations des buts :
ajb : pour ajouter un but dans buts
spb : pour supprimer un but dans buts
cbb : pour creer l ensemble buts
spbb : pour detruire tout buts
bb : pour imprimer buts

Commandes de manipulation de la base d'objets :
ajatt : pour ajouter une valeur a un attribut
afatt : pour affecter une valeur en ecrasant la precedente
supatt : pour supprimer une valeur a un attribut
modatt : pour modifier une valeur d'un attribut
obtatt : pour obtenir une valeur d'un attribut
defind : pour definir un individu
supind : pour detruire un individu
defsch : pour definir un concept
supsch : pour detruire un concept
modsch : pour modifier un concept
impression : pour imprimer un schema
visu : impression d'une instance
chind : pour rechercher les individus satisfaisant un filtre
spec : pour specialiser un individu
executer : pour activer un schema traitement
sortir : sauvgarde d'instances
lire : lecture et creation d'instances
lsch : liste des concepts
lind : liste des individus
supbind : supprimer tous les individus

supbsch : supprimer tous les concepts

Commandes de manipulations des regles :
pr : impression d une regle
prl : impression d une liste de regles
prc : impression de regles selon une recherche sur
sur le coefficient de vraisemblance
br : impression de la base de regles
mdr : pour modifier le texte d'une regle en edition de texte
mdrco : pour juste modifier le coeff de vraisemblance d'une regle
spr : suppression d une regle
ajr : ajout d une regle
rcp : pour generer la contraposee de regles
req : pour generer des regles equivalentes
spbr : supprimer toute la base de regles

compilation : pour compiler la base de regles
initialisation : pour initialiser la base compilee avec la
base de faits

Commandes de manipulations des fonctions :
ajfct : definir une nouvelle fonction interne ou en modifier
une existante
spfct : supprimer une fonction interne
prlfct : liste des fonctions internes creees

defext : definir une nouvelle fonction externe ou en modifier
une existante
spfext : supprimer une fonction externe
prlfext : liste des fonctions externes creees

Commandes de recherches parmi les regles :
rhp : recherches de regles sur un ou plusieurs mots dans
les premisses
rha : recherches de regles sur un ou plusieurs mots dans
les actions
rhfa : recherche d une forme dans les parties actions
rhfp : recherche d une forme dans les premisses
rht : recherche des regles selon leur type
rhi : recherche des regles d inference
rst : restriction de la base de regles utilisees

Commandes de manipulations des meta-regles :
pmr : impression d une meta-regle
bmr : impression de la base de meta-regles
ajmr : ajout d une meta-regle
spmr : suppression d une meta-regle
spbmr : suppression de toute la base de regles
mdmr : pour modifier une meta-regle
metam : pour activer 1 cycle du meta-moteur

Commandes de manipulations des indicateurs :
avecmeta : pour travailler en utilisant les meta-regles
sansmeta : pour ne plus utiliser les meta-regles
avecbutord : pour tenir compte de l ordre des buts partiels
sansbutord : pour ne pas tenir compte de leur ordre
rchex : pour utiliser une recherche exhaustive
nrchex : pour ne plus utiliser de recherche exhaustive
avectrace : travailler avec traces
sanstrace : travailler sans traces - mode adopte par defaut
avecomment : pour utiliser les commentaires des regles
sanscomment : pour ne plus utiliser les commentaires des regles
avecsauve : pour dupliquer le flot de sortie dans le
fichier criquet_dupl.ll
finsauve : pour annuler la commande avecsauve

Commandes pour activer un cheminement :
chav : activer le raisonnement en chainage avant
charr : activer le raisonnement en chainage arriere
charr-p : activer le raisonnement en chainage arriere partiel
chm : activer le raisonnement en chainage mixte

Commande de generation de regle (apprentissage) :
appr : pour generer une regle a partir du dernier
raisonnement tenu

Commande pour choisir une structure de controle :
chx : pour choisir la strategie de choix

Commande pour avoir les explications du systeme :
expl : pour avoir les explications sur le raisonnement

Commandes de sauvegardes et chargements :
sauvbrc : sauvegarde automatique sans parametre de la
BR courante
sauvbfc : sauvegarde automatique sans parametre de la
BF courante
resbr : restaurer l ancienne base de regles
resbf : recupere la derniere base de faits

sauvbf : sauvegarder la base de faits courante
sauvbr : sauvegarder la base de regles courante
sauvbmr : sauvegarder la base de meta-regles courante
sauvbb : sauvegarder la base de buts courante
sauvbh : sauvegarder la base d'hypotheses courante
sauvbo : sauvegarder la base d'objets courante

brs : pour lister le contenu d un fichier de regles
bfs : pour lister un fichier de faits

chargbf : charger une nouvelle base de faits
chargbr : charger une nouvelle base de regles
chargbmr : charger une nouvelle base de meta-regles
chargbb : charger une nouvelle base de buts

```
chargbh : charger une nouvelle base d'hypotheses
chargbo : charger une nouvelle base d'objets

pnbf : lister les noms utilises pour sauvegarder des BF
pnbr : lister les noms utilises pour sauvegarder des BR
pnbmr : lister les noms utilises pour sauvegarder des BMR
pnbh : lister les noms utilises pour sauvegarder des bases
       d'hypotheses
pnbb : lister les noms utilises pour sauvegarder des bases
       de buts
pnbo : lister les noms utilises pour sauvegarder des bases
       d'objets

Commandes diverses :
spnom : supprimer un nom  dans l'une des listes citees
ajnom : ajouter un nom dans une des listes citees

sauver : avec 1 param, sauvegarde dans un fichier Multics
charger : avec 1 param, charger un fichier Multics
edit : avec 1 param, acces en edition pleine page sur un fichier

avecsauve : pour dupliquer le flot de sortie dans le fichier
      criquet_dupl.ll
finsauve : pour arreter l'effet de la commande precedente

ruf : pour faire la reunion d un fichier de faits avec la
      base de faits courante
rur : pour faire la reunion d un fichier de regles avec la
      base de regles courante , utilisable aussi avec des meta-regles

end : pour sortir du systeme
```

PLAN

1) Définition d'un schéma
2) Définition d'une instance
3) Accès aux valeurs d'attributs
4) Manipulation des valeurs d'attributs
   Gestion de cohérence
5) Spécialisation d'une instance
6) Filtrage dans la base d'instances
7) Impression par niveaux
8) Suppression de schémas
9) Manipulation de schémas dans les règles

1/ Définition d'un schéma

```
CRIKA: lsch
s1 positif personne interroger compter_annees date

CRIKA: defsch
DEFINITION D'UN NOUVEAU SCHEMA

(salarie
   sorte_de:
      $valeur personne;
   salaire:
      $un entier
      $defaut 0);

CRIKA: impression salarie 2

     { salarie
          sorte_de
               $valeur  personne
          salaire
               $un  entier
               $defaut  0
     }

CRIKA: lsch
salarie s1 positif personne interroger compter_annees date

CRIKA: obtatt personne spec
schema : personne ; attribut : spec ; $valeur : salarie
```

2/ Définition d'une instance avec
---
résolution de références imbriquées

```
CRIKA: impression salarie 2
     { salarie
          sorte_de
               $valeur  personne
          salaire
               $un  reel
               $defaut  0.
     }

CRIKA: defind
DONNEZ UN NOM DE SCHEMA  : salarie
Schema salarie :
Nom de l'instance   ~ sinon: personne_2
salaire: ~
nom: "francois"
age: ~
date_naiss: >
          Schema date :
          Nom de l'instance   ~ sinon: date_2
          jour: 29
          mois: "mai"
          annee: 1951
a_pour_pere: personne_1
pere_de: ~
grand_pere_de: ~

CRIKA: lind
personne_2 date_2 g272 g271 g264 g263 g262 s1i g261 personne_3
personne_1 date_3 date_1

CRIKA: obtatt salarie inst
schema : salarie ; attribut : inst ; $valeur : personne_2
```

3/ Accès aux valeurs d'attributs

Différentes techniques d'inférences

```
CRIKA: impression salarie 1
     { salarie
          sorte_de
               $valeur  personne
          salaire
               $un  reel
               $defaut  0.
          inst
               $valeur  personne_2
     }

CRIKA: impression personne 2
     { personne
          lui_meme
               $variable  =lui
          nom
               $un  chaine
          age
               $un  entier
               $intervalle  0 120
               $variable  =age
               $si_besoin
               { compter_annees_265
                    origine
                         $variable  &date
                    annees
                         $variable  ?age
                    sorte_de
                         $valeur  compter_annees
               }
               { interroger_266
                    question
                         $valeur  age de la personne
                    reponse
                         $variable  ?age
                    sorte_de
                         $valeur  interroger
               }

          date_naiss
               $un  date
               $variable  =date
          a_pour_pere
               $un  personne
```

```
        pere_de
              $liste_de  personne
              $valeur
              { personne_267
                    a_pour_pere
                          $variable  &lui
                    sorte_de
                          $valeur  personne
              }

        grand_pere_de
              $liste_de  personne
              $valeur
              { personne_268
                    a_pour_pere
                          $valeur
                          { personne_269
                                a_pour_pere
                                      $variable  &lui
                                sorte_de
                                      $valeur  personne
                                 }

                    sorte_de
                          $valeur  personne
              }

        spec
              $valeur  salarie
        inst
              $valeur  personne_3 personne_1
  }
```

Accès direct

```
CRIKA: obtatt personne_2 nom
schema : personne_2 ; attribut : nom ; $valeur : francois
```

Accès avec inférence procédurale

```
CRIKA: obtatt personne_2 age
schema : personne_2 ; attribut : age ; $valeur : 34
```

Accès par filtrage

```
CRIKA: obtatt personne_1 grand_pere_de
schema : personne_1 ; attribut : grand_pere_de ; $valeur : personne_3
```

4/ Manipulation des valeurs d'attributs

Gestion de cohérence

```
CRIKA: impression date 2

     { date
          jour
               $un  entier
               $intervalle  1 31
          mois
               $un  chaine
               $domaine  janvier fevrier mars avril mai juin juillet
                         aout septembre octobre novembre decembre
          annee
               $un  entier
               $intervalle  1900 2000
          inst
               $valeur  date_3 date_2 date_1
     }
```

Utilisation de la facette $domaine

```
CRIKA: modatt date_2 mois "mai" "juilet"
erreur : valeur inconnue ou non admissible
```

Utilisation de la facette $intervalle

```
CRIKA: afatt date_2 jour 33
erreur : valeur deja presente ou non admissible

CRIKA: impression date_2 1

     { date_2
          est_un
               $valeur  date
          jour
               $valeur  29
          mois
               $valeur  mai
          annee
               $valeur  1951
     }
```

5/ Spécialisation d'une instance

```
CRIKA: impression personne_1 1
     { personne_1
          est_un
               $valeur  personne
          nom
               $valeur  roger
          date_naiss
               $valeur  date_1
     }

CRIKA: spec
DONNEZ UN NOM D'INDIVIDU  : personne_1
DONNEZ UN NOM DE SCHEMA OBJECTIF  : salarie

CRIKA: impression personne_1 1

     { personne_1
          est_un
               $valeur  salarie personne
          nom
               $valeur  roger
          date_naiss
               $valeur  date_1
     }

CRIKA: obtatt salarie inst
schema : salarie ; attribut : inst ; $valeur : personne_1 personne_2
```

6/ Filtrage dans la base d'instances

```
CRIKA: impression personne_3 1

     { personne_3
          est_un
               $valeur  personne
          nom
               $valeur  antoine
          date_naiss
               $valeur  date_3
          a_pour_pere
               $valeur  personne_2
     }

CRIKA: defsch
DEFINITION D'UN NOUVEAU SCHEMA

(filtrep
     sorte_de:
          $valeur personne;
     nom:
          $valeur "antoine";)


CRIKA: chind
DONNEZ UN NOM DE FILTRE A UTILISER  : filtrep
INDIVIDUS VERIFIANT LE FILTRE :
personne_3
```

7/ Impression par niveaux d'une instance

```
CRIKA: impression personne_3 3

     { personne_3
          est_un
               $valeur
               { personne
                    lui_meme
                         $variable  =lui
                    nom
                         $un  chaine
                    age
                         $un  entier
                         $intervalle  0 120
                         $variable  =age
                         $si_besoin  compter_annees_265
                              interroger_266

                    date_naiss
                         $un  date
                         $variable  =date
                    a_pour_pere
                         $un  personne
                    pere_de
                         $liste_de  personne
                         $valeur  personne_267

                    grand_pere_de
                         $liste_de  personne
                         $valeur  personne_268

                    spec
                         $valeur  filtrep salarie
                    inst
                         $valeur  personne_3 personne_1
               }

          nom
               $valeur  antoine
          date_naiss
               $valeur
               { date_3
                    est_un
                         $valeur  date
                    jour
                         $valeur  16
                    mois
                         $valeur  mai
                    annee
                         $valeur  1978
               }
```

```
a_pour_pere
      $valeur
      { personne_2
            est_un
                  $valeur  salarie
            a_pour_pere
                  $valeur  personne_1
            date_naiss
                  $valeur  date_2
            nom
                  $valeur  françois
      }

}
```

8/ Suppression de schémas

Gestion des liens verticaux

```
CRIKA: lsch
s1 positif salarie personne interroger compter_annees date

CRIKA: lind
g262 s1i g261 personne_3 personne_2 personne_1 date_3 date_2 date_1

CRIKA: supsch salarie
```

Le système détruit aussi les instances de "salarie"

Et met à jour le méta attribut "spec" dans "personne"

```
CRIKA: lind
g262 s1i g261 personne_3 personne_1 date_3 date_2 date_1

CRIKA: obtatt personne spec
schema : personne; attribut : spec ; $valeur : ()

CRIKA: lsch
s1 positif personne interroger compter_annees date
```

Même chose pour le schéma "personne"

```
CRIKA: obtatt personne inst
schema : personne ; attribut : inst ; $valeur : personne_3 personne_1

CRIKA: supsch personne

CRIKA: lind
g262 s1i g261 date_3 date_2 date_1

CRIKA: lsch
s1 positif interroger compter_annees date
```

9/ Manipulation de schêmas dans les règles

```
CRIKA: impression personne_2 1
     { personne_2
          est_un
               $valeur  salarie
          nom
               $valeur  francois
          date_naiss
               $valeur  date_2
          a_pour_pere
               $valeur  personne_1
          salaire
               $valeur  8000.
     }
```

Définition d'une règle

```
CRIKA: ajr
DONNEZ LE TYPE DE REGLE ?
PAR DEFAUT : i POUR INFERENCE
eq POUR EQUIVALENCE
: i
DONNER UN CONDITION , POUR FINIR ~ SEUL
: il-existe $un personne ?p
DONNEZ LE FILTRE A UTILISER
(filtrep
     sorte_de:
        $valeur personne;
     nom:
        $valeur "francois";)

: sachant ?s = ?p.salaire
DONNER UNE CONDITION , POUR FINIR ~ SEUL
: predicat : ?s > 7500.0
DONNER UNE CONDITION , POUR FINIR ~ SEUL
: ~
DONNER UNE ACTION , POUR FINIR ~ SEUL
: ajoutfait : salaire ?p.nom ?p.salaire
DONNER UNE ACTION , POUR FINIR ~ SEUL
: fonction : impression (quote ?p) 1
DONNER UNE ACTION , POUR FINIR ~ SEUL
: ~
DONNEZ DES COMMENTAIRES SI VOUS LE DESIREZ
: ~
DONNEZ DES REGLES CONSEILLEES POUR LA SUITE
SI VOUS LE DESIREZ
: ~
DONNER LE COEFFICIENT DE VRAISEMBLANCE
: 1
```

VOILA COMMENT J'AI COMPRIS CETTE REGLE

```
REGLE inference NUMERO 1
SI IL EXISTE $un ?p TEL QUE
(filtrep
     sorte_de:
          $valeur salarie ;
     nom:
          $valeur "francois" ;)

ET sachant ?s = ?p.salaire
ET predicat (?s > 7500.)
ALORS ajoutfait (salaire ?p.nom ?p.salaire)
ET fonction (impression '?p 1)
COEFF : 1

ETES VOUS D ACCORD ? oui - non
: oo
VOULEZ-VOUS CONTINUER ? oui-non
 n

CRIKA: br
REGLE inference NUMERO 1
SI IL EXISTE $un ?p TEL QUE
(filtrep
     sorte_de:
          $valeur salarie ;
     nom:
          $valeur "francois" ;)

ET sachant ?s = ?p.salaire
ET predicat (?s > 7500.)
ALORS ajoutfait (salaire ?p.nom ?p.salaire)
ET fonction (impression '?p 1)
COEFF : 1
```

Cheminement avant avec utilisation des schémas

```
CRIKA: chav
DEBUT CHEMINEMENT AVANT
SI VOUS RECHERCHEZ UN BUT PARTICULIER
DONNEZ SON TEXTE SINON RENVOYEZ ~ SEUL
: ~
     { personne_2
          est_un
               $valeur  salarie
          nom
               $valeur  francois
          date_naiss
               $valeur  date_2
          a_pour_pere
               $valeur  personne_1
          salaire
               $valeur  8000.
     }

RESULTATS :
BASE DE FAITS
==========================
salaire francois 8000.  COEFF : 1.
```

Explications

```
CRIKA: expl 2
 LE SYSTEME VA EXPLIQUER SON RAISONNEMENT A L AIDE DES
 REGLES UTILISEES
 LORS D UN CHEMINEMENT AVANT
==========================================
AVEC LA REGLE : 1
REGLE inference NUMERO 1
SI IL EXISTE $un personne_2 TEL QUE
(filtrep
     sorte_de:
          $valeur salarie ;
     nom:
          $valeur "francois" ;)

ET sachant 8000. = 8000.
ET predicat (8000. > 7500.)
ALORS ajoutfait (salaire francois 8000.)
ET fonction (impression 'personne_2 1)
COEFF : 1

ET LA BASE DE FAITS :
BASE DE FAITS
=============================

NOUS POUVONS DEDUIRE
ajoutfait (salaire francois 8000.)
fonction (impression 'personne_2 1)

---------------------------------
```

Allen E
"YAPS : a production rule system meets objects"
AAAI- 83, Aout 22-26 1983
Washington DC (p 5-7)

Bensaid A
"Un modèle de données relationnel étendu pour la mise
en oeuvre de bases de connaissances centrées objets"
Thèse de Docteur Ingénieur
Institut National Polytechnique de Grenoble Mai 1984

a/ Bobrow DG, winograd T
"An overview of KRL, a Knowledge Representation Language "
Cognitive Science Vol1 1977

b/ Bobrow DG, Kaplan RM, Kay M, Norman DA, Thompson H, Winograd T
"GUS : A Frame-Driven Dialog system"
Artificial Intelligence 8, 1977, (p 155-173)

Bonnet A
"L'utilisation des langages orientés objets (LOOBS) en
Intelligence Artificielle "
Les systèmes experts et leurs applications
Journées d'étude et exposition
Avignon 2,3 et 4 Mai 1984

Brachman RJ
"What ISA is and isn't. An analysis of taxonomic links in semantic
networks "
Computer Octobre 1983
IEEE Computer Society

Delobel C, Adiba M
"Bases de données et systèmes relationnels "
DUNOD 1982

Goldberg A, Robson D
"Smalltalk 80. The language and its implementation "
Addison Wesley Publishing company 1983

Goldstein I, Robert B
"Using Frame in scheduling "
in Artificial Intelligence
an MIT perspective, volume 1
Winston and Brown editors
MIT Press 1980

Goodwin J
"Taxonomic programming with KL-One "
Tech Rapport LiTH-MAT-R-79-5
Informatics Laboratory
Linkoeping University,Sweden 1979

Granger C,Thonnat M,Vignard P
"Etude d'un système expert pour la classification de galaxies : SYGAL"
Les systèmes experts et leurs applications
Journées d'étude 2,3 et 4 Mai 1984 Avignon

Halleux D
"Elaboration de maquettes de systèmes experts
appliquées au bâtiment"
Rapport PRC/444 juillet 1985
Centre Scientifique et Technique du Bâtiment
Division productique et construction
Etablissement de Sophia Antipolis

Hayes P
"The logic of frames"
in Reading in Artificial Intelligence (p 451-458)
Ed Webber BL, Nilson NJ
Tioga Publishing Company Palo Alto California 1981

Kuipers BJ
"A frame for frame : representation knowledge for recognition"
in "Representing and understanding"
Bobrow BG,Collins A (ed)
Academic Press 1975

Minsky M
"A framework for representing knowledge"
in "The psychology of computer vision"
Ph Winston Eds, Mc Graw Hill 1975

Pinson S
"Représentation des connaissances dans les systèmes experts "
RAIRO Informatique Vol 15,n=4,1981, (p 343-367)

Rechenmann F,Bensaid A,Granier D
"SHIRKA : des systèmes experts centrés objets"
Les systèmes experts et leurs applications
Journées d'étude et exposition
Avignon 2,3 et 4 Mai 1984

Rechenmann F
"Shirka : mécanismes d'inférence sur une base de connaissances
centrée objets"
Sième Congrès AFCET
Reconnaissance des Formes et Intelligence Artificielle
Grenoble 25-29 Novembre 1985

Thonnat M
"Automatic morphological description of galaxies
and classification by an expert system"
Rapport de Recherche INRIA n=387 Mars 1985

Vignard P
"Un logiciel de base pour élaborer des systèmes experts : CRIQUET"
Colloque International d'Intelligence Artificielle
Marseille 24-27 Octobre 1984

a/ Vignard P
"CRIQUET : un outil de base pour construire des systèmes experts"
Rapport de Recherche n 380 INRIA Mars 1985

b/ Vignard P
"Un mécanisme d'exploitation à base de filtrage flou pour
une représentation des connaissances centrée objets"
Thèse de 3ième cycle
Institut National Polytechnique de Grenoble, Juin 1985

c/ Vignard P
"Représentations de connaissances,
mécanismes d'exploitation et d'apprentissage"
Document INRIA Sophia Antipolis, 190 p
Support de cours INRIA (à paraître)

d/ Vignard P
"CRIQUET : manuel d'utilisateur"
Notes Internes INRIA Sophia Antipolis
Septembre 1985 (70 pages)

Winograd T
"Frame representation and the declarative/procedural controversy"
in "Representation and understanding"
Bobrow DG, Collins A (ed), Academic Press 1975

Winston PH, Horn BKP
"LISP"
Addison Wesley Publishing Company 1981

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique